A Complexity Theory viewpoint on the Software Development Process and Situational Context

Paul Clarke
School of Computing
Dublin City University, Ireland
Lero - Irish Software Research Centre
+353-1-700-7021
Paul.M.Clarke@.dcu.ie

Rory V. O'Connor
School of Computing
Dublin City University, Ireland
Lero - Irish Software Research Centre
+353-1-700-5643
Rory.OConnor@dcu.ie

Brian Leavy
DCU Business School
Dublin City University, Ireland
+353-1-700-5387
Brian.Leavy@dcu.ie

ABSTRACT

The research literature informs us that a software development process should be appropriate to its software development context but there is an absence of explicit guidance on how to achieve the harmonization of a development process with the corresponding situational context. Whilst this notion of harmonization may be intuitively appealing, in this paper we argue that interaction between a software development process and its situational context is an instance of a complex system. In Complexity Theory, complex systems consist of multiple agents that interact in a multitude of diverse ways, with system outcomes being non-deterministic. Complex systems are therefore noted to be difficult to control, such as is the case with many software development endeavors. If the interaction of software processes with situational contexts is representative of a complex system, then we should not be surprised that the task of software development has proven so resistant to attempts to produce generalized software processes. We should also seek to ameliorate the software development challenge through the adoption of techniques recommended for use in managing complex systems, not as a replacement for the many software process approaches presently in use, but as complement that can aid the task of process definition and evolution.

CCS Concepts

• Software and its engineering → Software creation and management → Software development process management → Software development methods.

Keywords

Software Development Process; Software Development Context; Complexity Theory; Software Process Optimization.

1. INTRODUCTION

Many in the software engineering field have suggested that no single software development process is perfectly suited to all software development settings [1]. This being the case, some amount of process adaptation, sometimes referred to as process tailoring [2], is required in order to render a process suitable to its environment. The environment has various factors that affect the software development process and for the purpose of this work, we refer to this collection of factors as the *situational context*. Given the inevitability of change in situational contexts, process adaptation should not be considered as a discrete, once off event.

Rather, adaptation is a continuous and complex activity, as it is not just the situational context that changes, but process innovations also emerge and interact with the situational context with the result that the challenge is unremitting and multidimensional. Indeed, it has been observed that when the richness of the software process is aligned with the devilish detail that exists in the situational context, the task of harmonizing a process with a context is in fact vast and beyond our ability to completely control [3]. Daunting though that observation me be, we simply cannot escape the reality that the software process is a continuous rather than a static concern [4] and so we should seek to identify techniques that can improve our understanding of interactions between software processes and their situational contexts.

Many different approaches to software development have been proposed, each claiming to offer something special or unique or new that represents an improvement to its antecedents, and in many cases new software process approaches (or at least many of those that gain acceptance across the community) do represent the potential for advancement (though it would seem that not all new offerings impart genuine newness, perhaps just new labels for pre-existing process concepts [5]). The very existence of many software development approaches suggests that a relatively high level of complexity may exist in software development. Indeed the very nature of software development, often creating something new based on a mere concept that requires reification in executable code, is characterized by a degree of uncertainty [6] that itself resonates with the central theme of a complex system (sometimes referred to as a complex adaptive system [7]). It also resonates with one of the central principles of effective complex adaptive systems, the law of requisite variety. Perhaps therefore, it should not come as a surprise to discover that the process used to produce software (and its interaction with its situation context) is also characterized by complexity and variety.

Prior to elaborating on the substantive details of this paper, it is worth briefly clarifying what the authors intend by the term *software development process* (or *software process* for short). Our preference is to adopt a simple and long-established definition which states that the *software process* is "the sequence of steps required to develop or maintain software" [8]. And although more recent software process innovations have not always identified themselves as a *process*, for example many approaches aligned with the Agile Manifesto [9] refer to themselves as *methods* or *methodologies* [10], [11], they remain congruent with our preferred

ICSSP'16, May 14-15, 2016, Austin, TX, USA

http://dx.doi.org/10.1145/2904354.2904369

definition – in that they represent a sequence of steps required to develop or maintain software. We therefore adopt the term *software process* in a very general sense, while also acknowledging that there are potentially varying views on this matter across the community.

In Section 2, we examine the level of complexity that exists in software processes, while section 3 reports on earlier work analyzing situational contexts. In section 4, the nature of complex systems is discussed further, and their relationship to software development is examined. Finally, a conclusion is presented in Section 5.

2. SOFTWARE PROCESS COMPLEXITY

Software development is undoubtedly a complex undertaking [12] that is beset with some difficult challenges. We know this because although many solutions to the software development process have been proposed [13], still the majority of software projects fall short of being completely successful though thankfully, success rates are reported to be improving over time [14]. Such is the variety of different software development approaches that is it not possible to offer a full critique herein, nor is that necessary as the objective is merely to discuss the general complexity surrounding software development processes rather than analyzing their specific composition. The explicit complexity in various software development lifecycle models (which are abstract representations of the process [15]) can vary, an observation that is especially evident if we view lifecycle models as being dichotomous, composed of *traditional* and *agile* software development lifecycle models.

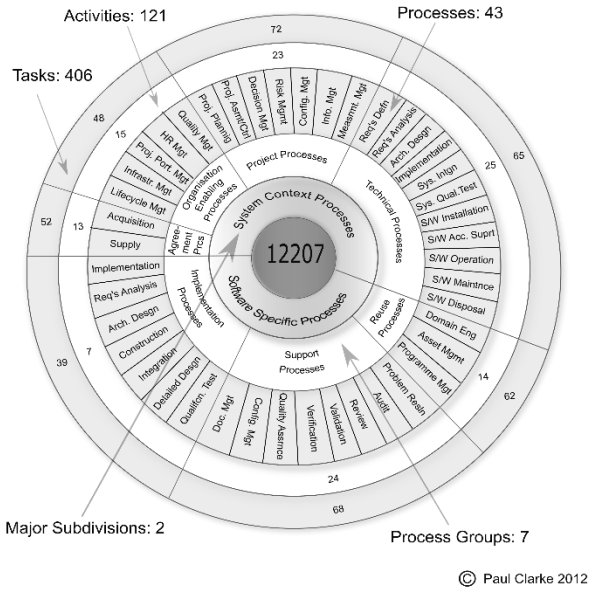

Figure 1. ISO/IEC 12207 Overview

Traditional models can be considered to comprise of long-established techniques such as the waterfall model [16] and the V-model [17], while quality management systems (QMS) (such as ISO 9001 [18]) and capability maturity frameworks (CMF) (such as CMMI [19]) are also dating from the era of traditional software process approaches. Of interest from a complexity theory perspective, large software process initiatives such as a CMF or a QMS carry with them relatively large and detailed process descriptions, offering evidence in support of the claim that a complete model of a complex process must necessarily be complex [20].

Taking ISO-33000 [21] as an example, the underlying software process lifecycle definition (ISO/IEC 12207 [15]) is presented as a hierarchy of processes, activities and task, with the lowest hierarchical level containing in excess of 400 individual tasks (refer to Figure 1). Broadly similar levels of process detail are to be found in CMMI, which must be applied to Level 3 in order to be eligible for various government contracts. One of the reasons that certain government departments may insist on CMMI Level 3 may be because the complex solution provided by CMMI has been shown to be effective in producing improved product quality and process predictability [22]; indicating that a complex solution can reduce the uncertainty associated with a complex system.

While CMFs have been shown to improve the consistency and predictability of software development, they may not be suited to all companies - and especially smaller companies - perhaps because they are costly or difficult to implement [23]. In contrast to traditional software development approaches, agile software development (based on the Agile Manifesto [9]) promotes the role of informal process implementation via self-organization and empowered teams that prefer interaction and discussion as a mechanism to address complexity rather than large, formal or bureaucratic process implementation. Therefore, although traditional and agile software development approaches are quite different in nature, they both tackle the complexity challenge (albeit it in different ways). And in the case of agile software development, there has been a noted application of some of the concepts from complexity theory in their design [24].

So we find that software development is complex irrespective of how the challenge is addressed, and the authors suggest that there may an underappreciation of this complexity in certain quarters that in itself may be impacting on the success or otherwise of software projects (and although success rates for projects have improved over time, there remains much room for further improvement [14]). To exemplify this view, all we need to do is reflect on the scenario where the non-software savvy executive pushes for deliverables which a software team must reify. Of course, it may be unreasonable to expect that those without detailed software development knowledge should fully appreciate the complexity involved (indeed, such is the gravity of the complexity problem that persons already equipped with detailed software development knowledge may themselves be unable to foresee the ramifications of their actions). The main point to emphasize here is that from the perspective of the arguments we present in this paper, the evidence overwhelmingly indicates that software development, governed by its software process, is inherently complex.

3. SITUATIONAL COMPLEXITY

While a great deal of material exists to demonstrate the complexity of software development processes, it appears that less attention has been focused on the area of software development situational contexts. This, the authors view to be somewhat surprising given the acknowledged importance of context in software process decisions [25]. Up until recently, and although it is noted that "the organization's processes operate in a business context that should be understood" [19] and that a "life cycle model… [should be] appropriate for the project's scope, magnitude, complexity, changing needs and opportunities" [15] , contributions to the software process context space may be lacking the level of detail that might be expected with a complex phenomenon.

A number of contributions have proposed various factors of the situation (or environment) that characterize the context of software development projects and of these contributions, it is earlier research from the authors of this work [26] that offers the most comprehensive list of situational factors presently published, owing to the fact that it is an accumulation of the factors evident in earlier contributions, including from areas such as software project risks, software cost estimation, software process tailoring, and assessing the degree of desirable process agility using the Boehm and Turner model.

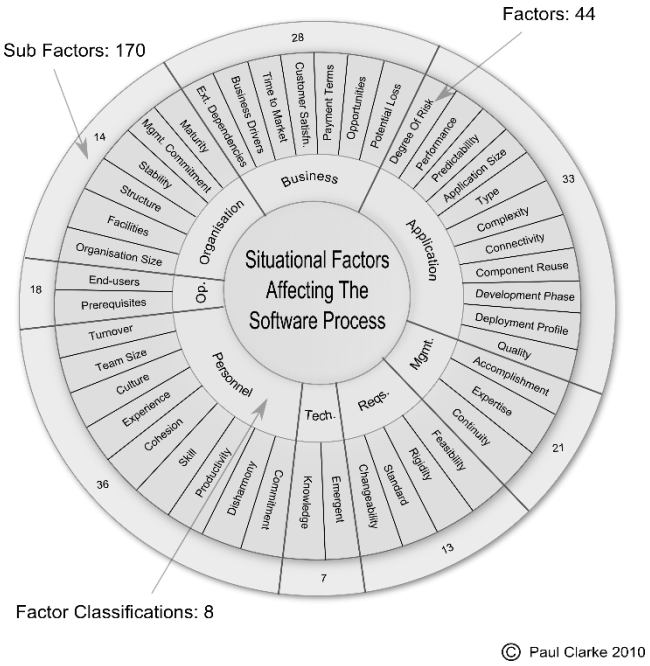

Figure 2. Situational Factors Affecting the Software Process

The situational factors framework [26] incorporates 44 individual factors affecting software development projects, which are further broken out into 170 sub-factors (refer to Figure 2). And although at this point in time the situational factors framework [26] may lack consensual validation, the number of distinct factors identified in the model serve to demonstrate that the software development situational context is a complex consideration. Furthermore, the trend has been towards the identification of increasingly larger reference frameworks for situational factors affecting software development, therefore it could be the case that as time progresses, even greater numbers of factors will be reported.

The evidence presented up to this point demonstrates that both the software process and its situational context are complex, and therefore we should expect their interaction to be of a complex nature. Concerns related to complex interactions in systems is the general focus of complex systems as described in complexity theory.

4. SOFTWARE DEVELOPMENT AS A COMPLEX SYSTEM

Although the primary interest of the authors lies in the software development field wherein established definitions of the term *system* already exist, other domains adopt the term *system* in different ways. In complexity theory [27], the term *system* identifies an object studied in some particular field, and such an object may be abstract or concrete, elementary or composite, linear or nonlinear, simple or complicated [28]. *Complex systems* can be considered to be highly composite, formed from large numbers of mutually interacting subunits whose interactions result in rich, collective behavior that feeds back into the behavior of the individual parts [28]. As such, a complex system is not constituted merely by the sum of its components, but also by the intricate relationships between these components [27]. Since they are continually changing, sometimes gradually and other times abruptly, complex systems may also be referred to as being *dynamic* [29].

This interpretation of the complex system concept can probably be considered analogous to terms already in use in describing the interaction between a software process and its context, such as *ambidexterity* [30] and *reflexivity* [31], which both refer to the phenomenon whereby software process subunits (sometimes referred to as process *activities* and *tasks* [15], other times as practices [32]) interact with various situational context subunits (sometimes referred to as *factors* [26]). Since a large number of software process and situational context subunits have been shown to exist, and seeing as their interactions are noted to be complex [12], it would appear that there may be benefits to examining complex systems research for utility in software development process management.

A further concept again, an *amethodical system* [33], has also been applied in the description of this type of interaction between a software process and its situational context, and in earlier related work the authors have accumulated some initial evidence from a longitudinal study that suggests that the capability to adapt a software process with respect to changing contexts is positively correlated with business success outcomes as viewed through the lens of an amethodical system [1]. Therefore, there is already an implicit awareness of complexity concepts in certain existing descriptions of the nature of the software process and its relationship with its situational context.

First proposed by the physical sciences and mathematics fields [29], the conceptual origins of complexity theory may be found in various domains including, philosophy of the organism [34] and neural networks [35]. Complexity theory is also closely related to the more general systems theory, especially with respect to viewing systems as a holistic set of interconnected elements [36]. And while advocates of complexity theory see it as a means of simplifying seemingly complex and dynamic systems [37], [38], there is however no single identifiable complexity theory. Rather, a number of theories concerned with complex systems gather under the general banner of complexity research [22], with a focus on examining how large numbers of elements or agents interact and give rise to high orders of complexity at a system level, with change being a central theme under consideration [29]. Change in complex systems is often non-linear, meaning that the effect is not proportionate to the cause [29].

It has been suggested that complexity theory consists of three distinct divisions: *Algorithmic*, *Deterministic* and *Aggregate* complexity [37] (indeed, in [37], Manson provides a review of complexity theory which the authors recommend to the novice reader and which is in effect summarized herein). Algorithmic complexity refers to the difficulty associated with describing system characteristics, with deterministic complexity focusing on the role of two or three key variables in creating largely stable systems such as in *chaos theory* and *catastrophe* theory [37]. Aggregate complexity attempts to access the *holism* resulting from the interaction of individual elements that work together to produce the apparently fluid harmony that characterizes complex systems,

and is therefore centrally concerned with the relationships between the constituent parts that comprise a complex system [37].

In this paper, we propose that the software process complex system be considered as an aggregate complexity concern wherein elements of the process interact with elements of the environment, a proposition which itself may be divergent from certain viewpoints concerning aggregate complexity that would appear to largely identify the environment as external to the complex system *per se* [37] (even if the complex system can influence the environment and vice-versa). This particular observation may ultimately represent an academic difference that has little impact on the potential benefits from applying the general concept of aggregate complexity to better understanding the interaction between a software process and its context. Furthermore, there have been earlier calls in the information systems (IS) domain to consider complexity theory as a frame of reference for IS design and evolution concerns [39], which are aligned with the general philosophy of agile software development in that there is an emphasis on enabling the evolution of a software product or system [24].

Although complex systems are necessarily complex and challenging to apply, it has been observed that software-based solutions – which are very much within our purview - are well suited to tackling complex systems, especially in the areas of modelling and simulation [37], [40]. It is also the case that previous work has examined the prospect of using modelling techniques to help software process design, for example in the 1990s there was some interest in software process modelling [41] (including work at the Software Engineering Institute [20]). However, earlier efforts at process modelling appear to not have been sustained and the authors of this paper have not identified much newly published material in this space in recent years. Perhaps it is the case that more recent information on the scope, complexity and content of software development contexts, coupled with the contemporary era of big data and data analytics, could reinvigorate efforts in this space, and as a community we could start to benefit from the theoretical benefits that modelling aggregate complexity may offer.

Among the benefits associated with complex systems concepts is the observation that firms that exhibit certain behaviors associated with complexity theory would appear to be deriving a competitive advantage. For example, comparisons of successful and less successful companies have shown that increased levels of success are witnessed where organizations maintain sufficient structure so as to avoid chaos, while at the same welcoming a degree adaptation and improvisation within projects [42]. Successful firms are also known to experiment with so-called *low cost probes* into the future [42], an example of which is new product speculation. Furthermore, successful companies seem more capable of linking the present with the future through process transition [42]. These types of examples may hold a particularly strong resonance for the software development domain, where the pace of change can be high.

While many are familiar with the seminal contribution by Charles Darwin to the theory of evolution, some may not be aware that although he observed and documented what appeared to be a system of evolution through adaptation, he did not attempt to suggest the precise mechanisms which underpin this adaptation [43], [44] (one suspects because it presented as being the result of complex interactions in systems that are not easily accessible to our perception). Therefore, important and established as the complex systems concept may be, including contemporary recognition of the role of adaptation in firing the engines of change [45], it is the case that what is intuitively appealing and accessible from a theoretical perspective remains elusive in the applied sense. And so, we in the software development field can perhaps draw some solace from the fact that our difficulties in managing software processes and by extension software projects, themselves an instance of a complex system, are echoed throughout both time and (seemingly) unrelated disciplines. And we certainly should not be surprised to discover that the ostensibly simple proposition that a software process should be appropriate to its context is in practice revealed as being layered in complexity.

5. CONCLUSION

In this paper, we have demonstrated that both software processes and the context within which they operate are complex considerations. We have further examined the area of complexity theory, finding that there are strong parallels between the challenge of harmonizing a software process with its context and the core challenges associated with complex systems as described in complexity theory. The complex system challenge is one that can be tackled through software-based modelling, which can help to evaluate the strengths of inter-relationships between entities. Complex systems have not gone unnoticed in the software field, with modelling having previously been adopted to support software processes [27], while more recently, the agile software development movement [13] has harnessed the power of self-organization which is a noted property of effective complex adaptive systems and a technique for addressing complexity as it arises in such systems. The development of a software-based model that can be trained with data from the practice of software development is an example of one possible technique that could be adopted in the examination of the benefits of aggregate complexity.

While the proposition that a software development process should be appropriate to its context is not likely to meet with much opposition, the discussion presented herein demonstrates that the application of this proposition is fraught with complexity. So great is this complexity that the challenge of harmonizing a software development process with its context may be underappreciated in some quarters, perhaps also by some software development professionals. However, many who have grappled with the challenges associated with software development will have a sense for the complexity involved and over the decades, many of the approaches proposed for software development have incorporated provisions to make their processes more resilient to the vicissitudes of the outside world. The effect of this paper has been to explicitly identify the resonance between complexity theory and the software development process. Approaches to tackling complex adaptive systems might be beneficial for software development processes.

REFERENCES

[1] P. Clarke, R. O'Connor, B. Leavy and M. Yilmaz. "Exploring the Relationship between Software Process Adaptive Capability and Organisational Performance," *IEEE Transactions on Software Engineering*, vol. 41, no. 12, pp. 1169-1183, 2015.

[2] G. Coleman and R. O'Connor. "Investigating software process in practice: A grounded theory perspective," *Journal of Systems and Software*, vol. 81, no. 5, pp. 772-784, 2008.

[3] T. Dyba. "Contextualizing Empirical Evidence," *IEEE Software*, vol. 30, no. 1, pp. 81-83, 2013.

[4] B. Curtis. "Three problems overcome with behavioral models of the software development process," *11th International Conference on Software Engineering*, pp. 398-399, 1989.

[5] N. Abbas, A. M. Gravell and G. B. Wills. "Historical roots of agile methods: Where did "Agile thinking" come from?" *Agile Processes in Software Engineering and Extreme Programming*, pp. 94-103, 2008.

[6] F. P. Brooks. "No Silver Bullet Essence and Accidents of Software Engineering," *Computer*, vol. 20, no. 4, pp. 10-19, 1987.

[7] K. B. Boal and P. L. Schultz. "Storytelling, time, and evolution: The role of strategic leadership in complex adaptive systems," *The Leadership Quarterly*, vol. 18, no. 4, pp. 411-428, 2007.

[8] W. S. Humphrey, *A Discipline for Software Engineering.* Reading, Massachusetts, USA: Addison-Wesley, 1995.

[9] M. Fowler and J. Highsmith, "The Agile Manifesto," *Software Development*, pp. 28-32, 2001.

[10] M. Lindvall, V. Basili, B. Boehm, P. Costa, K. Dangle, F. Shull, R. Tesoriero, L. Williams and M. Zelkowitz. "Empirical findings in agile methods," *Extreme Programming and Agile Methods — XP/Agile Universe 2002*, pp. 197-207, 2002.

[11] J. A. Highsmith, *Agile Software Development Ecosystems.* Boston: Addison-Wesley, 2002.

[12] A. Fuggetta. "Software process: A roadmap," *Proceedings of the Conference on the Future of Software Engineering*, pp. 25-34, 2000.

[13] R. Pressman, *Software Engineering a Practitioner's Approach.* Boston, MA: McGraw-Hill, 2005.

[14] The Standish Group International, Inc., *Chaos Manifesto 2013 - Think Big, Act Small.* Boston, MA: The Standish Group, 2013.

[15] ISO/IEC, *ISO/IEC 12207-2008 - Systems and Software Engineering – Software Life Cycle Processes.* Geneva, Switzerland: ISO, 2008.

[16] W. Royce, "Managing the development of large software systems: Concepts and techniques," *Western Electric show and Convention Technical Papers*, 1970.

[17] P. Rook. "Controlling software projects," *Software Engineering Journal*, vol. 1, no. 1, pp. 7-16, 1986.

[18] ISO, *ISO 9001:2000 - Quality Management Systems - Requirements.* Geneva, Switzerland: ISO, 2000.

[19] SEI, *CMMI for Development, Version 1.3.* CMU/SEI-2006-TR-008. Pittsburgh, PA, USA: Software Engineering Institute, 2010.

[20] W. S. Humphrey and M. I. Kellner, *Software Process Modeling: Principles of Entity Process Models. Technical Report CMU/SEI-89-TR-002, ESD-89-TR-002.* Pittsburgh, PA: Software Engineering Institute, 1989.

[21] ISO / IEC, *ISO/IEC 33000 Series on Information Technology - Process Assessment.* Geneva, Switzerland: ISO, 2015.

[22] D. E. Harter and S. A. Slaughter. "Quality Improvement and Infrastructure Activity Costs in Software Development: A Longitudinal Analysis," *Management Science*, vol. 49, no. 6, pp. 784-800, 2003.

[23] M. Staples, M. Niazi, R. Jeffery, A. Abrahams, P. Byatt and R. Murphy. "An exploratory study of why organizations do not adopt CMMI," *Journal of Systems and Software*, vol. 80, no. 6, pp. 883-895, 2007.

[24] K. Schwaber, "SCRUM development process," in *Business Object Design and Implementation*, , J. Sutherland, C. Casanave, J. Miller, P. Patel and G. Hollowell, Eds. . London: Springer, pp. 117-134, 1997.

[25] P. Feiler and W. Humphrey, *Software Process Development and Enactment: Concepts and Definitions.* CMU/SEI-92-TR-004. Pittsburgh, Pennsylvania, USA: Software Engineering Institute, Carnegie Mellon University, 1992.

[26] P. Clarke and R. V. O'Connor. "The situational factors that affect the software development process: Towards a comprehensive reference framework," *Journal of Information and Software Technology*, vol. 54, no. 5, pp. 433-447, 2012.

[27] P. Cilliers, *Complexity and Postmodernism - Understanding Complex Systems.* London, UK: Routledge Ltd., 1998.

[28] D. Rickles, P. Hawe and A. Shiell, "A simple guide to chaos and complexity," *Journal of Epidemiology and Community Health*, vol. 61, no. 11, pp. 933-937, 2007.

[29] D. Larsen-Freeman, *Chaos/Complexity Theory for Second Language Acquisition, in: The Encyclopedia of Applied Linguistics.* Hoboken, NJ: Blackwell Publishing Ltd, 2012.

[30] N. P. Napier, L. Mathiassen and D. Robey, "Building contextual ambidexterity in a software company to improve firm-level coordination ," *European Journal of Information Systems*, vol. 20, no. 6, pp. 674-690, 2011.

[31] R. V. O'Connor and P. Clarke. "Software process reflexivity and business performance: Initial results from an empirical study," *Proceedings of the 2015 International Conference on Software and System Process*, pp. 142-146, 2015.

[32] K. Schwaber, *Agile Project Management with Scrum.* WP Publishers & Distributors Pvt Limited, 2004.

[33] D. Truex, R. Baskerville and J. Travis. "Amethodical systems development: the deferred meaning of systems development methods," *Accounting, Management and Information Technologies*, vol. 10, no. 1, pp. 53-79, 2000.

[34] A. N. Whitehead, *Science and the Modern World.* New York: McMillan, 1925.

[35] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115-133, 1943.

[36] L. von Bertalanffy, *General Systems Theory and Psychiatry.* University of Michigan, MI: Little, Brown & Co., 1969.

[37] S. M. Manson. "Simplifying complexity: a review of complexity theory," *Geoforum*, vol. 32, no. 3, pp. 405-414, 2001.

[38] P. Anderson. "Perspective: Complexity Theory and Organization Science," *Organization Science*, vol. 10, no. 3, pp. 216-232, 1999.

[39] H. Benbya and B. McKelvey. "Toward a complexity theory of information systems development," *Information Technology & People*, vol. 19, no. 1, pp. 12-34, 2006.

[40] N. Thrift. "The Place of Complexity," *Theory, Culture & Society*, vol. 16, no. 3, pp. 31-69, 1999.

[41] M. I. Kellner, R. J. Madachy and D. M. Raffo. "Software process simulation modeling: Why? What? How?" *J. Syst. Software*, vol. 46, no. 2–3, pp. 91-105, 1999.

[42] S. L. Brown and K. M. Eisenhardt. "The Art of Continuous Change: Linking Complexity Theory and Time-Paced Evolution in Relentlessly Shifting Organizations," *Adm. Sci. Q.*, vol. 42, no. 1, pp. 1-34, 1997.

[43] D. Zohary, "Unconscious selection and the evolution of domesticated Plants ," *Economic Botany*, vol. 58, no. 1, pp. 5-10, 2004.

[44] C. Coulston Gillispie, "Lamarck and Darwin in the history of science," *American Scientist*, vol. 46, no. 4, pp. 388-409, 1958.

[45] H. Fineberg, "Are we ready for neo-evolution?" http://www.ted.com/talks/harvey_fineberg_are_we_ready_for_neo_evolution.html. 2012.